SYSTEMS ENGINEERING

Three-Step method to tightly integrate data mining tasks into a relational database system

INGENIERÍA DE SISTEMAS

Método Tres-Pasos para integrar fuertemente tareas de minería de datos en un sistema de base de datos relacional

§ Ricardo Timarán-Pereira

Departamento de Sistemas, Facultad de Ingeniería, Universidad de Nariño,
San Juan de Pasto, Colombia
§ ritimar@udenar.edu.co

(Recibido: 22 de Abril de 2103-Aceptado: 15 de Julio de 2013)

Abstract

In this paper, a result of the research project that aimed to define new algebraic operators and new SQL primitives for knowledge discovery in a tightly coupled architecture with a Relational Database Management System (RDBMS) is presented. In order to facilitate the tight coupling and to support the data mining tasks into the RDBMS engine, the three-step approach is proposed. In the first step, the relational algebra is extended with new algebraic operators to facilitate more expensive computationally processes of data mining tasks. In the next step and with the aim that the SQL language is relationally complete, these operators are defined as new primitives in the SELECT clause. In the last step, these primitives are unified into new SQL operator that runs a specific data mining task. Applying this method, new algebraic operators, new SQL primitives and new SQL operators for association and classification tasks were defined and were implemented into the PostgreSQL DBMS engine, giving it the capacity to discover association and classification rules efficiently.

Keywords: *Three-Step Method, Tight Coupling, Data Mining Tasks, Relational Database Management System.*

Resumen

En este artículo se presenta uno de los resultados del proyecto de investigación cuyo objetivo fue definir nuevos operadores algebraicos y nuevas primitivas SQL para el Descubrimiento de Conocimiento en una arquitectura fuertemente acoplada con un Sistema Gestor de Bases de Datos Relacional (SGBDR). Se propone el método tres-pasos con el fin de facilitar el acoplamiento fuerte y soportar tareas de minería de datos al interior del motor de un SGBDR. En el primer paso, se extiende el álgebra relacional con nuevos operadores algebraicos que faciliten los procesos computacionales más costosos de las tareas de minería de datos. En el siguiente paso y con el fin de que el lenguaje SQL sea relacionalmente completo, estos operadores son definidos como nuevas primitivas SQL en la cláusula SELECT. En el último paso, estas primitivas son unificadas en un nuevo operador SQL que ejecuta una tarea específica de minería de datos. Aplicando este método, se definieron nuevos operadores algebraicos, nuevas primitivas y operadores SQL para las tareas de Asociación y Clasificación y fueron implementados al interior del motor del SGBD PostgreSQL, dotándolo de la capacidad para descubrir reglas de asociación y clasificación eficientemente.

Keywords: *Método Tres-pasos, Acoplamiento Fuerte, Tareas de Minería de Datos, Gestor de Base de Datos Relacional.*

1. Introduction

Researches on data mining were initially concentrated on defining new patterns of discovery operations and developing algorithms for them. Subsequent researches (Agrawal and Shim, 1996), (Meo et al.,1998a), (Sarawagi et al., 2000), (Netz et al.,2000) have been focused on issues related to integrating data mining with database systems, producing as a result the systems and tools development of data mining whose architectures can be classified in one of three categories: loosely coupled, mildly coupled and tightly coupled with a database management system (DBMS) (Timarán, 2001).

Most data mining systems are loosely coupled with a DBMS. In this architecture, data mining algorithms are found outside the kernel of the DBMS. Integration is provided through an interface which function, in most cases, is limited to the commands "read from" and "write to" (Imielinski and Virmani, 1999). Their main disadvantages are poor scalability and performance. The first one arises when large data sets do not fit into the available memory and cannot therefore be mined efficiently. Poor performance arises when records are carried from the database address space to the application address space (Chaudhuri, 1998). To solve these problems, mining algorithms should be integrated into the DBMS engine as a primitive in a tightly coupled architecture (Timarán, 2001), (Boulicaut &Masson, 2010).

Many approaches to implement this kind of systems have been proposed. Expressing certain data mining operations as a series of SQL queries (Thomas & Chakravarthy, 1999), (Sarawagi et al., 2000),(Yoshizawa et al., 2000), (Rantzau,2004); extending SQL language with unified operators which support certain pattern discovery tasks: *DMQL* (Han et al., 1996) , *M-SQL* (Imielinski and Virmani, 1999), *MINE RULE* (Meo et al.,1998b); and, defining SQL generic primitives which facilitate the knowledge discovery process without supporting a particular task: NonStop SQL/MX primitives (Clear et al., 1999), *Count by Group* primitive (Freitas and Lavington, 1997), *FilterPartition, ComputeNodeStatistics* and *PredictionJoin* primitives (Sattler and Dunemann,2001).

A major drawback of the first approach of integration is poor performance, due mainly to the fact that the rather simple SQL operations like join, group and aggregation are not sufficient for efficiently executing data mining tasks (Sarawagi et al., 2000). One of the most important approaches to efficiently support the knowledge discovery in databases is to extend a DBMS engine with new operators and primitives. Meo et al. (1998a, 1998b) propose a unifying model to discover association rules. The model is based on a new operator, named *MINE RULE*, designed as an extension of the SQL language with a formal semantics for this operator. The semantics is described by means of an extended relational algebra with new operators: *Group by, Unnest, Extend, Substitute, Rename, Powerset*, which transform a relational table into an object-relational table (i.e. table with multivalued attributes) in order to discover association rules. *MINE RULE* is supported by tightly coupled architecture, where data mining is integrated within a classical SQL server. The differences between this approach and the proposed approach in this paper, is that the former does not propose SQL primitives that could be used in other discovery tasks. On the other hand, the new proposed algebraic operators conserve the closure property of the relational model and use relations with atomic attributes.

In (Clear et al., 1999) the implementation of a set of new SQL primitives: *Transpose, Vertical Partitioning, Round-robin, Horizontal Partitioning, sequence functions, sampling*, which were added to NonStop SQL/MX, a parallel, object-relational DBMS from the Tandem Division of Compaq; is reported. These primitives, along with other high-performance features of the SQL/MX engine enable basic knowledge discovery tasks to be performed in a scalable, efficient and parallel manner. Therefore, this type of integration is a very specific solution to a tightly coupled problem, since others that are not parallel to the DBMS could possibly not use these primitives. Also, these primitives do not have a formal definition in the relational algebra like the proposed primitives.

In this paper, one of the results of the research project that aimed to define new algebraic operators and new SQL primitives for knowledge discovery in a tightly coupled architecture with a Relational Database Management System (RDBMS) is presented. In order to facilitate the tightly coupled and to support the data mining tasks into the RDBMS engine, the three-step approach is proposed. In the first step, the relational algebra is extended with new algebraic operators to facilitate more expensive computationally processes of data mining tasks. In the next step and with the aim that the SQL language is relationally complete, these operators are defined as new primitives in the SELECT clause. In the last step, these primitives are unified into a new SQL operator that runs a specific data mining task. Applying this method, new algebraic operators, new SQL primitives and new SQL operators for association and classification tasks were defined and were implemented into the PostgreSQL DBMS engine, giving it the capacity to discover association and classification rules efficiently.

The rest of the paper has been organized as follows: In section 2, the methodology used to provide PostgreSQL DBMS capacities to knowledge discovery is presented. In section 3, new relational algebraic operators and new SQL primitives for association and classification tasks are described. Finally, in section 4 the conclusions are presented.

2. Methodology

The current database systems are designed primarily to support business applications. The success of the SQL language is linked to the small number of enough primitives to support the vast majority of these applications. Unfortunately, these primitives are not sufficient to support the emerging family of new applications dealing with Knowledge Discovery in Databases (KDD).

To support the data mining tasks into the Relational Database Management System (RDBMS) engine, the tree-step approach is used. This approach facilitates the tight coupling with a DBMS. In the first step, the relational algebra is extended with new operators that execute the most expensive processes of association and classification tasks. In the second step, SQL language is extended with new primitives in the SQL SELECT clause that implement the new relational algebraic operators. Finally, in the last step, the new SQL primitives are unified into new SQL operators, that allow the extraction of association and classification rules, in the new SQL clause.

3. Results and discussion

3.1 New operators of relational algebra for data mining tasks

A data mining architecture tightly coupled with DBMS, a new algebraic operator should execute the most expensive processes of data mining tasks to guarantee efficiency in the data mining operations.

For an association task, the overall performance of mining association rules is determined by the discovery of large itemsets, i.e., the sets of itemsets that have their support above a pre-determined minimum support (Han and Kamber, 2001). For a classification task by decision tree induction, a decision tree classifier is built in two phases: a growth phase and a pruning phase. In the growth phase, the tree is built by recursively partitioning the data until all members belong to the same class. The tree growth is computationally much more expensive than the pruning phase. In the first phase, to compute the attribute selection measure is the most expensive part of the algorithm since finding the best split for a node requires evaluating the attribute selection measure for each attribute at each possible split point (Wang et al, 1998).

3.1.1 Algebraic operators for an association task

The process of extracting association rules is facilitated extending the Relational Algebra with the following new operators proposed in (Timarán, 2005):

3.1.1.1 Associator (α)

The *Associator* operator *(α)* generates, for each tuple of the relation *R*, all their possible subsets (itemsets) of different size. The *Associator* takes each tuple *t* of *R* and two numbers *IS* and *ES* as input, and returns, for each tuple *t*, the different combinations of attributes Xi from size *IS* until size *ES*, as tuples in a new relation. In each tuple Xi, only the attributes that are combined have values, the rest of attributes are null. The order of the attributes in the R scheme determines the order of the attributes in the subsets.

Formally, let $A = \{A_1, \ldots, A_n\}$ be the set of attributes of relation *R; n* and *m* are degree and cardinality of *R* respectively; *IS* y *ES* are the initial and final size of the subsets to obtain respectively:

$$\alpha(IS; ES; R) = \{ \cup_{all} X_i \mid X_i \subseteq t_i, \forall_i \forall_k (X_i = <v_i(A_1), v_i(A_2), null.., v_i(A_k), null>, \quad (i \leq (2^n - 1) * m), (k = IS..ES)), and\ A_1 < A_2 < \ldots < A_k \}$$

Example 1. Let R (A, B, C) be the relation in Figure 1a. Let R1 = α(2; 3; R) be the operation. The output of the Associator is shown in Figure 1b.

3.1.1.2 EquiKeep (χ)

EquiKeep (χ) is a unary algebraic operator that as the *Selection(σ)* operator, evaluates a logical expression from a relation R, but *EquiKeep* applies the logical expression to the columns (attributes) of R. This operator restricts the attribute values of each one of the tuples of a specified relation to only the attribute values that satisfy a specified condition, making the rest of attribute values null. *EquiKeep* takes each tuple *t* of a relation *R* and a logical expression *P* as input, and returns a new relation with the same R scheme, in which, each new tuple *t* is formed by the attribute values that satisfy the expression P. The rest of attribute values are made null. *EquiKeep* eliminates the empty tuples, i.e. the tuples with all the attribute values null.

Formally, let $A = \{A_1, \ldots, A_n\}$ be the set of attributes of relation *R; n* and *m*, degree and cardinality of *R* respectively. Let *P* be a logical expression:

$$\chi_p(R) = \{ t_i(A) \mid \forall i \forall j\ (p(v_i(A_j)) = v_i(A_j)\ \ if\ \ p = true\ and\ p(v_i(A_j)) = null\ and\ p = false),\ i = 1 \ldots m',\ j = 1 \ldots n,\ m' \leq m \}$$

Example 2. Let R (A, B, C) be the relation in Figure 1a. Let A=a1 v B=b1 v C=c2, the logical expression to evaluate. The result of R1=$\chi_{A=a1\ v\ B=b1\ v\ C=c2}$(R) is shown in Figure 1c.

3.1.1.3 Describe associator (βα)

A *Describe Associator* is a unary algebraic operator that takes as input the resulting relation of the *Associator* and for each tuple of this relation, it generates, from not null *l* attributes of the tuple, all the different subsets of specific size like {{a},{l-a},s}, where {a} is named antecedent subset and {l-a} consequent subset. Subsets {a} and {l-a} are subset of *l* attributes. The s is the size of the antecedent subset {a}.

Formally, let A = {A1, ..., An} be the set of attributes of relation R; *n* and *m* are degree and cardinality of *R* respectively. Let *LR* be the size of the subsets to obtain.

$$\beta\alpha_{LR}(R) = \{ \cup_{all} X_i(Y) \mid Y = \{Y_1, Y_2, .. Y_{LR}, S\}, X_i \subseteq t_i, t_i \in R, \forall_i\ (X_i = <v_i(A_1), .., v_i(A_k), s>, v_i(A_k) <> null), (i \leq (2^n - 2) * m), LR \leq n \}$$

The β operator applied to R produces a new relation with degree *LR+1*, cardinality $i \leq (2^n - 2)$ and the schema *R (Y), Y=* $\{Y_1, Y2, .. Y_{LR}, S\}$, where S is the length of antecedent subset. The *Describe Associator* facilitates the generation of one-dimensional or multidimensional association rules [HaKa01].

Example 3. Let R (A, B, C) be the relation in Figure 1a to obtain all subsets of size 3. The result of $\beta\alpha_3\,(R)$ is shown in Figure 1d.

3.1.2 Algebraic operators for classification task

The process of extracting classification rules is facilitated extending the Relational Algebra with the following new operators proposed by Timarán (2005):

3.1.2.1 Mate (μ)

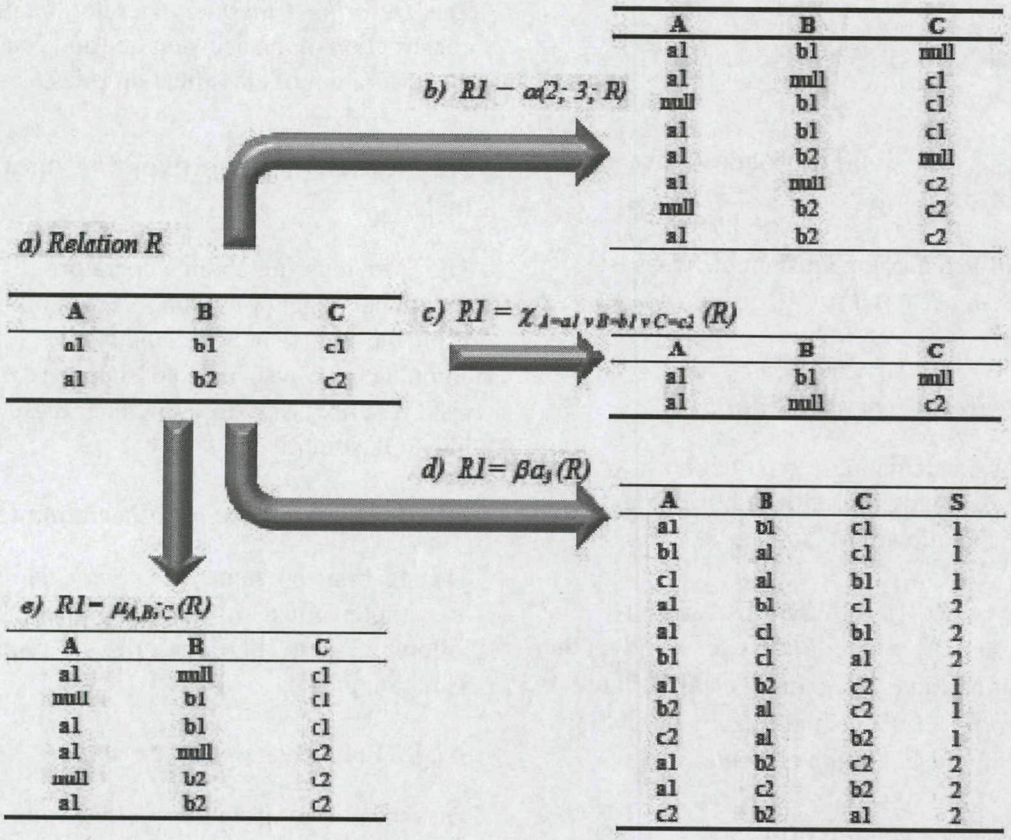
The *Mate* operator (μ) generates, for each tuple of relation *R*, all their possible combinations of the not null attribute values from an attributes list denominated *Condition Attributes*, with the not null *Class Attribute* value. This process is executed in a single passing on the relation.

Formally, let $A = \{A_1, \ldots, A_n\}$ be the set of attributes of relation *R*; *n* and *m* are degree and cardinality of *R* respectively; $LC \subset A$, $LC \neq \phi$ the *Condition Attributes* list and n' the size of LC, $|LC| = n', n' < n$. Let $Ac \in A$, $Ac \cap LC = \phi$ be the *Class Attribute*. The Mate operator (μ) is defined this way:

$\mu_{LC;\,Ac}(R) = \{\, ti(M) \mid M = LC \cup Ac,\ LC \subset A,\ |LC| = n', n' < n,\ Ac \in A,\ Ac \cap LC = \phi,\ ti = Xi,\ 1 \leq i \leq m',$

$m' = (2^{n'}-1)*m,\ \forall_i \forall_k (\ X_i = <null, \ldots, v_i(A_k) \ldots, null, \ldots, v_i(Ac)>,\ v_i(A_k)\, v_i(Ac) \neq null),\ 1 \leq k \leq n' \}$

Example 4. Let R (*A, B, C*) be the relation in figure 1a. Let R1= $\mu_{A,B;C}$(R) be the operation. The output of Mate is shown in Figure 1e.



a) Relation R

A	B	C
a1	b1	c1
a1	b2	c2

b) R1 − α(2; 3; R)

A	B	C
a1	b1	null
a1	null	c1
null	b1	c1
a1	b1	c1
a1	b2	null
a1	null	c2
null	b2	c2
a1	b2	c2

c) R1 = $\chi_{A=a1 \vee B=b1 \vee C=c1}$ (R)

A	B	C
a1	b1	null
a1	null	c2

d) R1= $\beta\alpha_3$ (R)

A	B	C	S
a1	b1	c1	1
b1	a1	c1	1
c1	a1	b1	1
a1	b1	c1	2
a1	c1	b1	2
b1	c1	a1	2
a1	b2	c2	1
b2	a1	c2	1
c2	a1	b2	1
a1	b2	c2	2
a1	c2	b2	2
c2	b2	a1	2

e) R1− $\mu_{A,B;C}$ (R)

A	B	C
a1	null	c1
null	b1	c1
a1	b1	c1
a1	null	c2
null	b2	c2
a1	b2	c2

Figura 1. *Algebraic operators. a) Relation R b)Output of Associator operator c) Output of EquiKeep operator d) Output of Describe Associator operator e) Output of Mate operator*

3.1.2.2 Aggregate operator Entro

The *Entro* allows to calculate the entropy measure of a relation R with regard to a condition attribute and a class attribute.

Formally, let $A = \{A_1, \ldots, A_n\}$ be the set of attributes of relation R; n and m are degree and cardinality of R respectively. Suppose the class attribute Ac, $Ac \in R(A)$, has t distinct values defining t distinct classes, C_i $(1 \le i \le t)$. Let r_i be the number of tuples of R *in* Ci class. Let q be the number of distinct values $\{v_1(A_k), v_2(A_k), \ldots, v_q(A_k)\}$ of condition attribute A_k, $A_k \in R(A)$, which can be used to partition R into q subsets $\{S_1, S_2, \ldots S_q\}$, where S_j contains those tuples in R that have a value $v_j(A_k)$ of attribute A_k. Let s_{ij} be the number of tuples of C_i class in a subset Sj. $Entro(A_k; Ac; R)$, return the entropy of R regarding attribute A_k, in this way:

$Entro(A_k;Ac; R) = \{y \mid y = -\sum p_{ij} \log_2(p_{ij}),\ 1 \le i \le t,\ 1 \le j \le q,\ p_{ij} = s_{ij} / |S_j|\}$

where $pij = s_{ij} / |S_j|$ is the probability that a tuple in S_j belongs to C_i class.

Entropy of R regarding attribute *Ac* class is:
$Entro\ (Ac;Ac; R) = \{y \mid y = -\sum p_i \log_2(p_i),\ 1 \le i \le t,\ p_i = r_i / m\}$

3.1.2.3 Aggregate operator Gain

Gain allows calculating the reduction in entropy caused by knowing the value of attribute A_k. Gain is defined as following:

$Gain\ (A_k;Ac; R) = \{y \mid y = Entro(Ac;Ac; R) - Entro(A_k;Ac; R)\}$ where $Entro\ (Ac; Ac; R)$ is the entropy of relation R regarding class attribute Ac and $Entro\ (A_k; Ac; R)$ is the entropy of relation R regarding the condition attribute A_k.

3.1.2.4 Describe classifier (βμ)

A *Describe Classifier* (βμ) is a unary operator that takes the resulting relation of the operators *Mate, Entro and Gain* as input, and returns a new relation with the attribute values that will form the different nodes of the decision tree.

Formally, let $A = \{A_1, .., A_n, E, G\}$ be the set of attributes of relation R; $n+2$ and m are degree and cardinality of R respectively. The *Describe Classifier* (βμ) operator is defined this way:
$\beta\ (R) = \{\ t_i(Y) \mid Y = \{N,P,A,V,C\}\ \}$

where,

$t_i = <val(N), null, val(A), null, null>$ if t_i is a root node,

$t_i = <val(N), val(P), val(A), val(V), val(C)>$ if t_i is a leaf node

$t_i = <val(N), val(P), val(A), val(V), null>$ if t_i is other node

The *Describe Classifier* operator facilitates the construction of the decision tree and consequently the generation of classification rules.

3.2 New SQL primitives for data mining tasks

The previous algebraic operators extend the Relational Algebra for support data mining tasks. With the aim that SQL language is relationally complete and also able to support data mining task, it is necessary to implement these operators like SQL primitives.

3.2.1 SQL primitives for association task

The algebraic operators *Associator,* and *EquiKeep* are implemented in SQL language with the following new SQL primitives proposed by Timarán (2005):

3.2.1.2 **Primitive *associator range***

The primitive *Associator Range* in the SQL SELECT clause implements the Algebraic operator *Associator*. In the SELECT clause, this primitive has the following syntax:

SELECT *<AttributeListDataTable>* [INTO *<AssociatorTableName>*]
FROM *<DataTableName>*
WHERE *<WhereClause>*
ASSOCIATOR RANGE*<number 1>* TO *<number 2>*
GROUP BY *<AttributeListAssociatorTable>*

The ASSOCIATOR RANGE clause determines the size of different subsets that are generated by this primitive, starting from an initial size < *number 1* >, until <TO> a final size < *number 2* >. The other clauses are standard SQL clauses and therefore their functions are very well known for all.

The ASSOCIATOR RANGE primitive facilitates the calculation of the large itemsets for discovery association rules in multicolumn tables (Rajamani et al., 1999).

Example 5. Let *Students* (PROGRAM, AGE, GENDER, STRATUM, AVERAGE) be table in figure 2a. Get large itemsets of size 2 and 3, formed by the PROGRAM, GENDER and STRATUM attributes with minimum support greater than or equal to 2 and store them in the table *AssoStudents*.

The SQL query is:
SELECT *program, gender, stratum, count(*)* AS *support* INTO A*ssoStudents*
FROM *Students*
ASSOCIATOR RANGE *2* UNTIL 3
GROUP BY *program, gender, stratum*
HAVING *count(*)>=*2

The final result of this query is shown in figure 2b.

3.2.1.2 Primitive *EquiKeep On*

The primitive *EquiKeep On* in the SQL SELECT clause implements the algebraic operator *EquiKeep*. In the SELECT clause, *EquiKeep On* has the following syntax:

SELECT*<AttributeListDataTable>*[INTO *<EquiKeepTableName>*]
FROM *<DataTableName>* WHERE *<WhereClause>*
EQUIKEEP ON < *Condition* >

EQUIKEEP ON < *Condition* > clause keeps the values of the attributes of the table *<AttributeListDataTable>*, maintaining in each record of the table *<EquiKeepTableName>* only the attribute values that satisfy a specified condition *<condition>*. The rest of attribute values of the table *<EquiKeepTableName>* become null. The primitive EQUIKEEP ON facilitates the generation of large itemsets in the discovery of Association Rules, to keep in each record of the table only the values of the frequent attributes.

Example 6. Keep in each record of the table *Students* of figure 2a, only the values of the attributes that satisfy the following conditions: PROGRAM like Systems or Languages, AGE like 21..25, GENDER like F, STRATUM like 2 or 4 and AVERAGE like regular or low. Store the result in the table *EquiStudents.*

The SQL statement is:
SELECT * INTO *EquiStudents*
FROM Students
EQUIKEEP ON *program* in ('Systems','Languages), *age* like'21..25', *gender* = 'F', *stratum* in (2,3), *average* in ('Regular','Low')

The result of this statement is shown in figure 2c

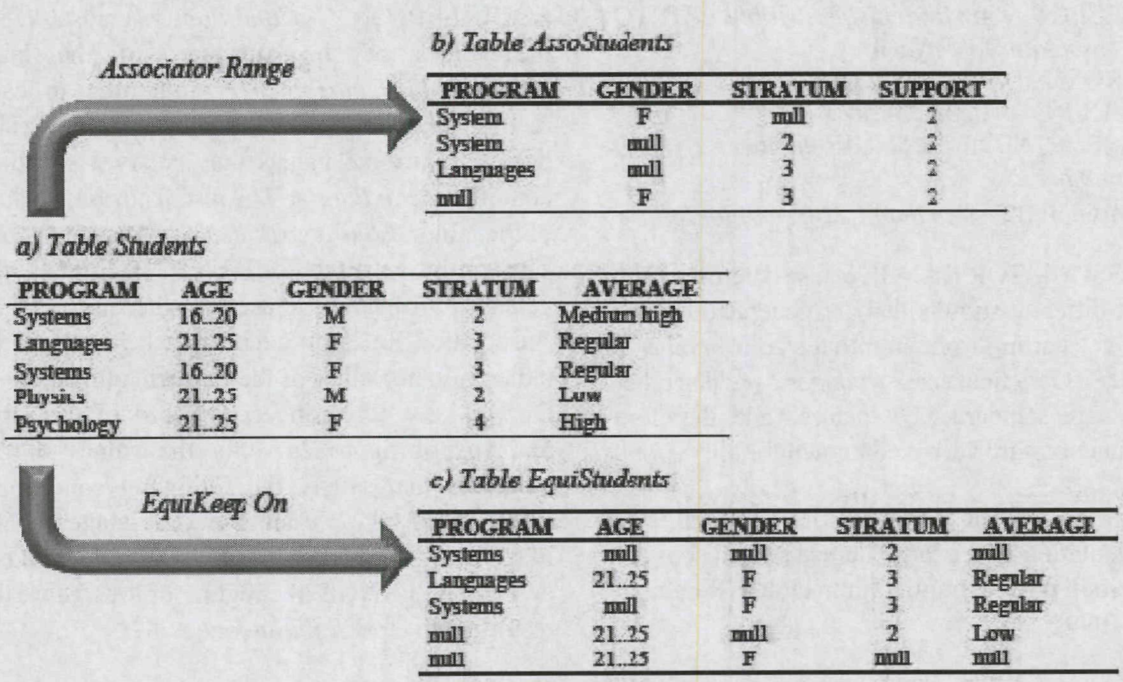
3.2.2 SQL Primitives for classification task

The algebraic operator *Mate,* together with the aggregate operators *Entro*() and *Gain*() are implemented in SQL language with the following new SQL primitives:

3.2.2.1 Primitive *mate by with*

The primitive *Mate by* in the SQL SELECT implements the algebraic operator *Mate.* This primitive has the following syntax in the SELECT clause:

SELECT *<AttributeListDataTable>* [INTO *<MateTableName>*]
FROM *<DataTableName>*
WHERE *<WhereClause>*
MATE BY*<ConditionAttributesList>* WITH *<ClassAttribute>*
GROUP BY < *AttributeListDataTable>*



b) Table AssoStudents

PROGRAM	GENDER	STRATUM	SUPPORT
System	F	null	2
System	null	2	2
Languages	null	3	2
null	F	3	2

a) Table Students

PROGRAM	AGE	GENDER	STRATUM	AVERAGE
Systems	16..20	M	2	Medium high
Languages	21..25	F	3	Regular
Systems	16..20	F	3	Regular
Physics	21..25	M	2	Low
Psychology	21..25	F	4	High

c) Table EquiStudents

PROGRAM	AGE	GENDER	STRATUM	AVERAGE
Systems	null	null	2	null
Languages	21..25	F	3	Regular
Systems	null	F	3	Regular
null	21..25	null	2	Low
null	21..25	F	null	null

Figura 2. *SQL Primitives for Association Task a) Table Student b)Result of Associator Range c) Result of EquiKeep On*

MATE BY<*ConditionAttributesList*> WITH <*ClassAttribute*> determines the set of attributes <*ConditionAttributesList*> with which the attribute class <*ClassAttribute*> is combined.

The primitive MATE BY facilitates the classification task and the construction of a decision tree. This primitive calculates together with aggregate functions Gain *()* and *Entro (),* in each partition and for each attribute, the information gain and entropy respectively.

Example 7. Let *Symptoms* (SID, PAIN, FEVER, INFLUENZA) be table in figure 3a. Perform different combinations between attributes PAIN and FEVER with attribute INFLUENZA, obtain their occurrences and store the result in the table *ClasSymptoms*.

The SQL command that performs this query is:

> SELECT *pain, fever, influenza, count(*)* AS support INTO *ClasSymptoms*
> FROM *Symptoms*
> MATE BY pain*, fever* WITH *influenza*
> GROUP BY *pain, fever, influenza*

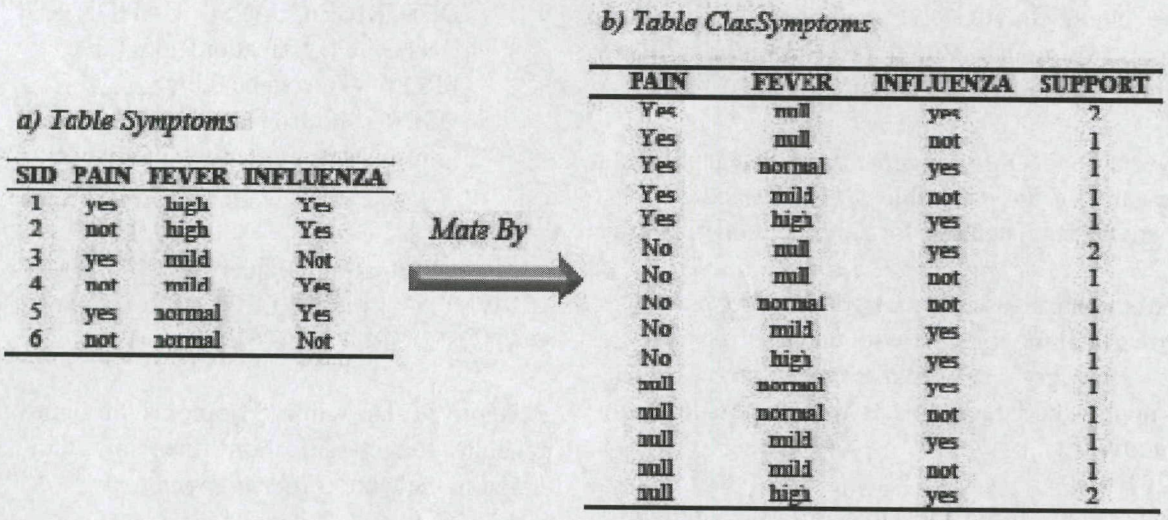
The result of this query is shown in figure 3b.

3.2.2.2 Aggregate function *Entro()*

The Algebraic aggregate operator *Entro()* is implemented by the aggregate function *Entro()* in the SQL SELECT. This function has the same syntax as the primitive MATE BY WITH:

> SELECT <*AttributeListDataTable*>, *Count(*), Entro(*)* [INTO <*MateTableName*>]
> FROM <*DataTableName*>
> WHERE <*WhereClause*>
> MATE BY<*ConditionAttributesList*>
> WITH <*ClassAttribute*>
> GROUP BY < *AttributeListDataTable*>

The aggregate function Entro () calculates, together with the primitive *Mate By with*, the entropy of each of the combinations of the condition attributes with the attribute class. SQL


a) Table Symptoms

SID	PAIN	FEVER	INFLUENZA
1	yes	high	Yes
2	not	high	Yes
3	yes	mild	Not
4	not	mild	Yes
5	yes	normal	Yes
6	not	normal	Not

Mate By

b) Table ClasSymptoms

PAIN	FEVER	INFLUENZA	SUPPORT
Yes	null	yes	2
Yes	null	not	1
Yes	normal	yes	1
Yes	mild	not	1
Yes	high	yes	1
No	null	yes	2
No	null	not	1
No	normal	not	1
No	mild	yes	1
No	high	yes	1
null	normal	yes	1
null	normal	not	1
null	mild	yes	1
null	mild	not	1
null	high	yes	2

Figura 3. *SQL Primitives for Classification Task a) Table Sympotms b)Result of Mate By*

Entro() must be run together with the aggregate function *count* ().

3.2.2.3 Aggregate function *Gain ()*

The Algebraic aggregate operator Gain () is implemented by aggregate function Gain () in the SQL SELECT. This function has the same syntax as primitive MATE BY WITH :

```
SELECT <AttributeListDataTable>, Count
(*), Entro(*), Gain(*)
[INTO <MateTableName>]
FROM <DataTableName>
WHERE <WhereClause>
MATE BY<ConditionAttributesList>
WITH <ClassAttribute>
GROUP BY < AttributeListDataTable>
```

The aggregate function Entro () calculates, together with the primitive *Mate By with*, the gain of information of each of the combinations of the condition attributes with the attribute class. SQL Gain () must be run together with the aggregate functions *count* () and *Entro*().

3.3 New SQL Operators for data mining tasks

The SQL language has been extended with primitives for Association and Classification tasks that are expressed in the SQL SELECT clause. These primitives facilitate the most computationally expensive processes of these tasks. Now, it is necessary to unify these primitives into SQL operators that allow extracting association and classification rules efficiently. These new operators are:

3.3.1 SQL Operator for association task

The SQL operator that unifies association primitives is called *Describe Association Rules*. This SQL operator implements the algebraic operator *Describe Associator* in a new SQL clause. The *Describe Association Rules* generates association rules with a specific length from large itemsets.

The operator *Describe Association Rules* has the following syntaxis:

```
DESCRIBE ASSOCIATION RULES INTO
<AssociationRulesTable>
FROM <LargeItemsetsTable>
WITH CONFIDENCE <valor1>
LENGTH <valor2 >
[DO <LargeItemsetsSubquery>]
< LargeItemsetsSubquery >::=<SFWEAG>
<SFWEAG> := <SELECT FROM WHERE
EQUIKEEP ASSOCIATOR GROUP BY>
```

The clause INTO < *AssociationRulesTable* > allows storing in a table < *AssociationRulesTable* > the association rules for future querys.

The clause FROM <*LargeItemsetsTable*> specifies the name of the data table <*LargeItemsetsTable*> where the large itemsets for the extraction rules are

The optional clause DO <*LargeItemsetsSubquery*> allows defining together with the *describe* clause, the subquery <*LargeItemsetsSubquery*> that computes the large itemsets with the association primitives.

Example 8. From the table *Students* of figure 2a, find the association rules of length 3 with a minimum support of 2 and a minimum confidence of 30. The generated rules are stored in the table *AssorulesStudents.*

The SQL command that performs this query is:

```
DESCRIBE ASSOCIATION RULES INTO
AssorulesStudents
FROM Assostudents
WITH CONFIDENCE 30 LENGTH 3
DO    SELECT program, gender, stratum,
count(*) AS support INTO Assostudents
FROM Students
EQUIKEEP    ON    program    in
('Systems','Languages), age like'21..25',
gender = 'F', stratum in (2,3), average in
('Regular','Low')
ASSOCIATOR RANGE 2 UNTIL 3
GROUP BY program, gender, stratum
HAVING count(*)>=2
```

3.3.2 SQL Operator for classification task

The SQL operator that unifies classification primitives is called *Describe Classification Rules*. This SQL operator implements the algebraic operator *Describe Classifier* in a new SQL clause. The *Describe Association Rules* builds the decision tree and generates classification rules.
The operator *Describe Classification Rules* has a similar syntaxis of Describe Association Rules:

```
DESCRIBE CLASSIFICATION RULES
[INTO <ClassificationRulesTable>]
FROM <TreeNameTable>
USING <MetricNameTable>
[DO  <MetricCalculationSubquery>]
```

where

< MetricCalculationSubquery >::=<SFWMG>
<SFWMG> ::= <SELECT FROM WHERE MATE BY GROUP BY>

Example 9. The table Symptoms of figure 3a, generate the classification rules and they are stored in the table *ClassrulesSymptoms.*

The SQL command that performs this query is:

```
     DESCRIBE CLASSIFICATION RULES
     INTO Classrulessymptoms
     FROM Treenodes
     USING Gainsymptoms
DO
     SELECT pain, fever, influenza, count(*),
     Entro(*), Gain(*) INTO Gainsymptoms
     FROM Symptoms
     MATE BY pain, fever WITH influenza
     GROUP BY pain, fever, influenza
```

In this example from *gainsymptoms* table, the operator *Describes Classification Rules* builds the table *treenodes* and with it generates the classification rules and stores them in the table *classrulessymptoms*.

3.4 Implementation of new SQL primitives and SQL Operators for data mining tasks

The new SQL primitives and new SQL operators for Association and Classification tasks were implemented into the engine of PostgreSQL DBMS. This process involved the modification of the structures, functions and the creation of new nodes in some components of the architecture of Postgres. The Parser was modified to build, transform and attach to the structures of the compiler a list with the new primitives and operators. The Planner / Optimizer was modified to receive the parser tree and recognize the new primitives and operators, in which case new nodes were added to Query plan. The Executor was modified to evaluate the new nodes and deliver a set of tuples, according to query. As a result of this process PostgresKDD, a database system with the capacity to support the discovery of association and classification rules in large data sets, was obtained.

4. Conclusions

The three-steps approach was applied to integrate in a tight coupling, the association and classification tasks into a relational database system. The Relational Algebra was extended with the new operators: *Associator*, *Equikeep, Describe Associator*, *Mate, Entro*, *Gain* and *Describe Classifier.* The SQL was extended with the primitives *Associator Range, EquiKeep On, Mate by* and the aggregate functions *Gain()* and *Entro()*. Also, SQL was extended with the operators *Describe Association Rules* and *Describe Classification Rules*. These primitives and operators for data mining tasks were implemented into PostgreSQL engine.

The future works in this area include following this method to define new algebraic operators and primitives for different data mining tasks and their implementation in PostgreSQL DBMS and extend the query optimizer of this DBMS, so that it executes a data mining query efficiently.

5. References

Agrawal R.& Shim, K. (1996). *Developing Tightly-Coupled Data Mining Applications on a Relational Database System.* In proceedings of The Second International Conference on Knowledge Discovery and Data Mining, Portland, Oregon, USA, p. 287-290.

Boulicaut, JF., Masson, C. (2010). *Data Mining Query Languages.* Data Mining and Knowledge Discovery Handbook. Second Edition, Springer, p. 655-664, ISBN: 978-0-387-09823-4.

Chaudhuri, S. (1998). Data Mining and Database Systems: Where is the Intersection?. *Bulletin of the Technical Committee on Data Engineering 21* (1), 4-8.

Clear, J., Dunn, D., Harvey, B., Heytens, M., Lohman, P., Mehta, A., Melton, M., Rohrberg, L., Savasere, A., Wehrmeister, R. & Xu, M. (1999). *NonStop SQL/MX Primitives for Knowledge Discovery.* In proceedings of KDD-99, San Diego, USA.

Freitas, A.,& Lavington, S. (1996). *Using SQL Primitives and Parallel DBServers to Speed Up Knowledge Discovery in large relational databases.* In proceedings of XIII European Meeting on Cybernetics and Systems Research, Vienna, Austria, p. 955-960.

Han, J., Fu, Y., Wang, W., Koperski, K. & Zaiane, O. (1996). *DMQL: A Data Mining Query Language for Relational Databases.* In proceedings of SIGMOD 96 Workshop, On research issues on Data Mining and Knowledge Discovery DMKD 96, Montreal, Canada.

Han, J., Kamber, M. (2001). *Data Mining Concepts and Techniques.* San Francisco, California, USA: Morgan Kaufmann Publishers.

Imielinski, T. & Virmani, A. (1999). MSQL: A Query Language for Database Mining. *Data Mining and Knowledge Discovery*, Kluwer Academic Publishers, 3 (4), 373-408.

Meo, R., Psaila G. & Ceri S. (1998a). *A Tightly-Coupled Architecture for Data Mining.* In proceedings of 14th. International Conference on Data Engineering ICDE98.

Meo R., Psaila G. & Ceri S. (1998b). An Extension to SQL for Mining Association Rules. *Data Mining and Knowledge Discovery,* Kluwer Academic Publishers, 2 (2), 195-224.

Netz, A., Chaudhuri, S., Bernhardt, J. & Fayyad U. (2000). *Integration of Data Mining and*

Relational Databases. In Proceedings of the 26thInternational Conference on Very Large Databases, Cairo, Egypt.

Rantzau, R. (2004). Frequent Itemset Discovery with SQL Using Universal Quantification. *Database Support for Data Mining Aplications, Lecture Notes in Computer Science*, Springer, 2682, 194-213.

Rajamani, K., Cox, A., Iyer, B. & Chadha, A.(1999). *Efficient Mining for Association Rules with Relational Database Systems*. In proceedings of International Database Engineering and Aplication Symposium, Montreal, Canada, p. 148-155.

Sarawagi, S., Thomas, S. & Agrawal, R. (2000). Integrating Association Rule Mining with Relational Database Systems: Alternatives and Implications. *Data Mining and Knowledge Discovery*, Kluwer Academic Publishers, 4 (2/3), 89-125.

Sattler, K & Dunemann, O.(2001). *SQL Database Primitives for Decision Tree Classifiers*. In proceedings of Conference on Information and Knowledge Management, Atlanta, Georgia, USA.

Timarán, R. (2001). Arquitecturas de Integración del Proceso de Descubrimiento de Conocimiento con Sistemas de Gestión de bases de datos: un Estado del Arte. *Revista Ingeniería y Competitividad,* Universidad del Valle, 3 (2), 45-55.

Timarán, R. (2005). *Nuevas Primitivas SQL para el Descubrimiento de Conocimiento en Arquitecturas Fuertemente Acopladas con un Sistema de Gestión de Bases de Datos*. Tesis doctoral, Doctorado en Ingeniería, Escuela de Ingeniería de Sistemas y Computación, Facultad de Ingenierías, Universidad del Valle, Cali, Colombia.

Thomas, S. & Chakravarthy, S. (1999). *Performance Evaluation and Optimization of Join Queries for Association Rule Mining*. In Proceedings of First International Conference on Data Warehousing and Knowledge Discovery , DAWAK.

Wang, M., Iyer, B. & Scott, V. (1998). *Scalable Mining for Classification Rules in Relational Databases*. In proceedings of International Database Engineering and Application Symposium, Cardiff, U.K., p. 58-67.

Yoshizawa, T., Pramudiono, I. & Kitsuregawa, M. (2000). SQL Based Association Rule Mining using Commercial RDBMS (IBM DB2 UDB EEE). *Data Warehousing and Knowledge Discovery,* Springer, 1874, 301-306.